Arithmetic Data Cube as a Data Intensive Benchmark

Michael A. Frumkin, Leonid Shabanov[1]
NASA Advanced Supercomputing (NAS) Division
NASA Ames Research Center, Moffett Field, CA 94035-1000
frumkin@nas.nasa.gov, leonid.shabanov@crossz.com

NAS-03-005

February 20 2003

[1]Employee of CrossZ Solutions USA.

Abstract

Data movement across computational grids and across memory hierarchy of individual grid machines is known to be a limiting factor for application involving large data sets. In this paper we introduce the Data Cube Operator on an Arithmetic Data Set which we call Arithmetic Data Cube (ADC). We propose to use the ADC to benchmark grid capabilities to handle large distributed data sets. The ADC stresses all levels of grid memory by producing 2^d views of an Arithmetic Data Set of d-tuples described by a small number of integer parameters. We control data intensity of the ADC by controlling the sizes of the views through choice of the tuple parameters.

1 Introduction

The main subject of data warehousing and *On-Line Analytical Processing* (OLAP), decision support database systems, data mining systems and resource brokers is a data set represented as a list of tuples. The data set has a number of dimension and measure attributes. Here we will consider data sets having d of dimension attributes and a single measure attribute. A tuple t of such data set can be represented as $t = (i_1, \ldots, i_d, c)$, where each dimension attribute i_j assumes values in some range, say in an interval $[1, m_j - 1]$, and c is a cost function (a measure) associated with t. The goal of OLAP is to assist users to discover patterns and anomalies in the data set by providing short query execution times [13].

A standard tool of OLAP is the *Data Cube Operator* (DCO) [3], which computes views of the data set. For a chosen subset of k attributes, a view is a sorted set of k-tuples containing only the chosen attributes with accumulated measures of the duplicates. If technically possible, DCO computes 2^d views on all possible subsets of the dimensions. The calculated DC reduces queries of multidimensional data to simple look-ups. There are approaches [1, 5] for mining multi-dimensional association rules and answering iceberg queries by computing an iceberg cube, which contains only aggregates above a certain threshold.

The input data sets and some of the materialized views often do not fit in-core, thus DCO computation requires a careful reuse of data loaded into the main memory (and all levels of cache). As a rule, computation of the DCO spills data across all levels of memory, making DCO especially interesting as a data intensive benchmark. Also, DCO output size usually is significantly larger than the size of the input data set.

A large number of papers is devoted to efficient computation of the DCO [6, 12, 18] and many companies have proprietary algorithms for DCO computations. Some authors propose parallel DCO computation algorithms [8, 10]. To improve the efficiency of querying data cubes a number of publications consider calculation and storage of data cubes as condensed cubes [17] or as other highly compressed structures [14].

We are not trying to evaluate DCO algorithms here, instead we are designing a benchmark for computational grids. For the reference implementation[1] we choose a greedy algorithm [6] that computes each view from the smallest parent (a view having one more attribute). We assume that all attribute values are integers. Although real OLAP data sets and existing OLAP benchmarks [11, 16] use mostly strings as attribute values, this is not a significant limitation, since strings can be enumerated by integers (using hashing, for example) if necessary. One of the advantages of using integers as attribute values is reduction in the size of the input data sets and materialized views.

Many data sets are available to test OLAP systems, DCO algorithms and data mining algorithms, for example, the ABP-1 and TPC-D benchmark databases [11, 16]. For benchmarking purposes the most appropriate is a synthetic data set which can be generated by a small program, so that the data set will be scalable, the distribution of the benchmark will be manageable, and replication of the data set on the computational grid will incur a small overhead. Also, a synthetic data set, as in many real applications, can be generated in a distributed fashion, which saves the effort of splitting and distributing the data set.

[1] To be described in our next report

In available synthetic data sets, the tuples are randomly generated, however there is no way to control the sizes of the data views. In the next section we introduce *the Arithmetic Data Set*, which is similar to the randomly generated data sets, but has the advantage of a priori known sizes of the views. The latter simplifies the implementation of the greedy DCO algorithm. For real or random data, one can estimate the view sizes using sampling or some analytical methods [6, 15].

2 The Arithmetic Data Set

The purpose of constructing an *Arithmetic Data Set* is to have a data set whose view sizes can be well controlled. An Arithmetic Data Set S is a subset of a group Q defined by

$$Q = \bigoplus_{i=1}^{d} (\mathbb{Z}/m_i\mathbb{Z})^*,$$

where $(\mathbb{Z}/m_i\mathbb{Z})^*$ is the set of integers modulo m_i coprime with m_i. An element of S can be represented by a tuple $x = (x_1, \dots, x_d)$, where x_i is an integer modulo m_i. The subset S is defined by a seed $s = (s_1, \dots, s_d) \in Q$, a generator $g = (g_1, \dots, g_d) \in Q$, $s_i, g_i \neq 0,\ i = 1, \dots, d$ and the total number of elements n:

$$S = \bigcup_{j=0}^{n-1} (s_1 g_1^j, \dots, s_d g_d^j).$$

We choose $1 < g_i < m_i$ to be one of $f_i = |(\mathbb{Z}/m_i\mathbb{Z})^*|$ numbers which are coprime with m_i. Let q_i be the *order* of g_i that is the smallest integer such that $g_i^{q_i} \equiv 1 \bmod(m_i)$. Since g_i^j can assume at most f_i different values then $g_i^{f_i} \equiv 1 \bmod(m_i)$ and q_i divides f_i. The tuples $(s_1 g_1^j, \dots, s_d g_d^j),\ j = 0, \dots, n-1$ are different elements of Q if $\mathrm{LCM}(q_1, \dots, q_d) \geq n$, see Corollary 2. (Here LCM stands for the Least Common Multiple.)

Data Views. For any subset containing k of the cube dimensions $I = \{i_1, \dots, i_k\} \subset \{1, \dots, d\}$ the I-view of $x \in Q$ is defined as a projection of x on the face of the cube defined by I:

$$x_I = (x_{i_1}, \dots, x_{i_k}).$$

The I-view of S is comprised of the I-view of all elements of S:

$$S_I = \{x_I | x \in S\}.$$

View Sizes. For a given I-view we are interested to find out the number of tuples in S_I. To do this we estimate *the multiplicity* of a tuple $x \in S_I$, defined as the number of tuples of S having the same I-view as x.

Two tuples $s_I g_I^j$ and $s_I g_I^k$ are the same iff $g_I^k = g_I^j$ or $g_I^{k-j} = 1_I$ considered as elements of Q_I. Hence, the multiplicity μ of $s_I g_I^j$ can be calculated as follows:

$$\mu = |\{0 \leq k < n \mid k - j \equiv 0 \bmod(q_i),\ i \in I\}|.$$

Since the smallest nonzero solution of the system of congruences $k - j \equiv 0 \bmod(q_i),\ i \in I$, is $\lambda_I = \mathrm{LCM}_{i \in I}(q_i)$ we find that $\lfloor \frac{n}{\lambda_I} \rfloor \leq \mu \leq \lfloor \frac{n}{\lambda_I} \rfloor + 1$, which proves the following assertion.

PROPOSITION 1. *Let $\lambda_I = LCM_{i \in I}(q_i)$. The multiplicity μ of any tuple of an I-view of S can be estimated as*

$$\lfloor \frac{n}{\lambda_I} \rfloor \leq \mu \leq \lfloor \frac{n}{\lambda_I} \rfloor + 1.$$

If $\lambda_I > n$, then the second inequality of the proposition implies that multiplicity of each element of S_I is 1, hence $|S_I| = n$. Obviously, $|S_I| \leq \lambda_I$. Hence, we have the following formula for $|S_I|$:

COROLLARY 2. *For the size of an I-view of S we have the following relation:*

$$|S_I| = min(n, \lambda_I).$$

3 Choice of the Parameters

To illustrate a possible choice of the parameters for the grid benchmarks we choose m_i to be prime numbers and g_i to be generators of $(\mathbb{Z}/m_i\mathbb{Z})^*$, hence having period $q_i = f_i = m_i - 1$. Also, we choose m_i such that $m_i - 1$ has many small prime factors so that λ_I has a good chance of been small. This approach gives us good control over the sizes of the data set and its views. Our actual choice of the m_i is shown in the Table 1.

We choose 4 groups of the smallest prime numbers $\{3, 5, 7\}$, $\{11, 13, 17, 19\}$, $\{23, 29, 31, 37\}$, and $\{41, 43, 47, 53, 59\}$. For each group we choose 5 smallest primes m_i such that prime factors of $m_i - 1$ are 2 and numbers from this group[2]. This set of parameters gives us a data set of $2^5 \cdot 3^2 \cdot 5^2 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19^2 \cdot 23 \cdot 29 \cdot 31^2 \cdot 37 \cdot 41 \cdot 43 \cdot 47 \cdot 53 \cdot 59$ different tuples and, for example we can choose $n = 2 \cdot 11 \cdot 23 \cdot 41 \cdot 3 \cdot 13 \cdot 29 \cdot 43 \cdot 5 \cdot 17 = 85759918530$. At the same time the sizes of 5-dimensional views (relative to each of the groups) are small relative to the number of the total elements in the data set.

4 DCO Application in Air Traffic Control

The Air Traffic Control (ATC) system works with records of flight data represented as a set of tuples [9]. We use ATC to illustrate a possible application of DCO to speedup access to the data.

Each of about 20 national ATC Centers obtain flight data from airports and radars in real time. Typical records are shown in Table 2 and a typical query is as follows:

```
Find AC type
  where  Busy = 1
  and ETA is Between 1105 and 1110
  and destination is CLE
```

These queries can be posted at any of the centers and should be executed in real time. It implies that:

- the flight data must be communicated between the centers in real time;
- at each center the data should be stored in the form that allow real time queries;

One possible way to satisfy the second requirement is to keep at each center complete data cube in core memory.

This example addresses only queries of ATC data. It shows that DCO can be used to process data sets distributed across nodes of a gird. A possibility of real time update of ATC data and of the Data Cube should be farther investigated.

[2]Since we use odd primes, $m_i - 1$ always has 2 as a factor

Table 1. **Dimensions of the Arithmetic Data Cube and their factorizations. Here "Least Generator" γ_i is the smallest generator of $(\mathbb{Z}/m_i\mathbb{Z})^*$, the "Generator" is the chosen generator of $(\mathbb{Z}/m_i\mathbb{Z})^*$ and the "Exp" is e_i such that $g_i = \gamma_i^{e_i}$.**

Prime	Factorization of $m-1$	Least Generator	Exp	Generator	Seed $(m+1)/2$
1. 421	$2^2 \cdot 3 \cdot 5 \cdot 7$	2	11	364	211
2. 601	$2^3 \cdot 3 \cdot 5^2$	7	13	412	301
3. 631	$2 \cdot 3^2 \cdot 5 \cdot 7$	3	17	334	316
4. 701	$2^2 \cdot 5^2 \cdot 7$	2	19	641	351
5. 883	$2 \cdot 3^2 \cdot 7^2$	2	23	108	442
LCM	$2^3 \cdot 3^2 \cdot 5^2 \cdot 7^2 = 88200$				
6. 419	$2 \cdot 11 \cdot 19$	2	23	228	210
7. 443	$2 \cdot 13 \cdot 17$	2	29	98	222
8. 647	$2 \cdot 17 \cdot 19$	5	31	94	324
9. 21737	$2^3 \cdot 11 \cdot 13 \cdot 19$	31	37	8280	10869
10. 31769	$2^3 \cdot 11 \cdot 19^2$	7	41	26667	15885
LCM	$2^3 \cdot 11 \cdot 13 \cdot 17 \cdot 19^2 = 7020728$				
11. 1427	$2 \cdot 23 \cdot 31^2$	2	41	595	714
12. 18353	$2^4 \cdot 31 \cdot 37$	3	43	8397	9177
13. 22817	$2^5 \cdot 23 \cdot 31$	3	47	15046	11409
14. 34337	$2^5 \cdot 29 \cdot 37$	3	53	15699	17169
15. 98717	$2^2 \cdot 23 \cdot 29 \cdot 37$	2	59	62206	49359
LCM	$2^5 \cdot 23 \cdot 29 \cdot 31^2 \cdot 37 = 758228608$				
16. 3527	$2 \cdot 41 \cdot 43$	5	3	125	1764
17. 8693	$2^2 \cdot 41 \cdot 53$	3	5	443	4347
18. 9677	$2^2 \cdot 41 \cdot 59$	2	7	128	4839
19. 11093	$2^2 \cdot 47 \cdot 59$	2	11	2048	5547
20. 18233	$2^3 \cdot 43 \cdot 53$	3	13	8052	9117
LCM	$2^3 \cdot 41 \cdot 43 \cdot 47 \cdot 53 \cdot 59 = 2072850776$				

5 Related Work

The benchmarking of data mining systems is well established area of High Performance Computing [11, 16]. These benchmarks are designed to compare performance of query systems running on a server. On the other hand, a number of benchmarks have been designed for testing computational grids [2]. The grid benchmarking effort is currently supported by the Grid Benchmarking Research Group at the Global Grid Forum. These benchmarks are mostly computationally intensive and are derived from NAS Parallel Benchmarks. We propose the Arithmetic Data Cube (ADC) as a data intensive grid benchmark which extends typical data mining operations into a grid environment.

6 Conclusions

We show that ADC represents an important set of computations in the OLAP and data mining. We give an example of a dynamic real time system performing the set of operations specified in ADC.

Table 2. **Air Traffic Control Data. Typical Query: `Find AC type where Busy = 1 and ETA is Between 1105 and 1110 and destination is CLE`.**

Flight ID	AC type	ETA	Destination	Controller	Busy
UAL 147	747	1100	CLE	17	1
NW 1186	767	1132	ORD	26	1
KLM 761	747	1105	CLE	8	1
AA 2345	A320	1135	ORD	17	1
UAL 258	737	1112	CLE	9	1
AA 2744	737	1105	CAK	11	1
SW 377	767	1108	CLE	87	1

The ADC is data intensive since

- it mostly involves logical operations
- the size of the output data set significantly exceeds the size of the original data set
- existing algorithms perform few operations per memory access (and are similar to the merge in this respect)

The advantages of ADC as a grid benchmark are that

- it is described by a small number of parameters and has a priori known sizes of the views
- the views can be generated independently
- the overhead of combining the generated views is predictable
- the data set can be partitioned into a number of independently generated subsets
- the elements of the data set are pseudo random

These two properties make ADC a strong candidate for a data intensive grid benchmark to be considered by the Global Grid Forum Grid Benchmarking Research Group (GB-RG) [4].

Bibliography

[1] K. S. Beyer and R. Ramakrishnan. *Bottom-up computation of sparse and iceberg cubes.* SIGMOD 1999, 359-370.

[2] M. Frumkin, Rob F. Van der Wijngaart. *NAS Grid Benchmarks: A Tool for Grid Space Exploration.* Cluster Computing, Vol. 5, pp. 247-255, 2002.

[3] J. Gray, A. Bosworth, A. Layman, and H. Prahesh. *Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total.* Microsoft Technical Report, MSR-TR-95-22, 1995.

[4] *Grid Benchmarking Research Group.* http://www.ggf.org/L_WG/wg.htm.

[5] J. Han, J. Pei, G. Dong, K. Wang. *Efficient Computation of Iceberg Cubes with Complex Measures.* SIGMOD'01, Santa Barbara, CA, May 2001, 1-12.

[6] V. Harinarayan, A. Rajaraman, and J. D. Ullman. *Implementing Data Cubes Efficiently.* In Proc. of ACM SIGMOD, pp. 205-216, Montreal, Canada, June 1996.

[7] *IBM Quest Synthetic Data Generation Code.* http://www.almaden.ibm.com/cs/quest/syndata.html.

[8] W. Liang, M. E. Orlowska. *Computing Multidimensional Aggregates in Parallel.* International Conference on Parallel and Distributed Systems, Taiwan, 1998, 92-99.

[9] W. Meilander, M. Jin, J. Baker. *Tractable Real-Time Air Traffic Control Automation.* Proceedings of the 14th IASTED International Conference Parallel and Distributed Computing and Systems, Cambridge, USA, 2002, pp. 483-488.

[10] S. Muto, M. Kitsuregawa. *A Dynamic Load Balancing Strategy for Parallel Datacube Computation.* Proceedings of the second ACM international workshop on Data warehousing and OLAP, 1999, 67-72.

[11] *OLAP Council / ABP-1 OLAP Benchmark, Release II,* http://www.olapcouncil.org.

[12] S. Sarawagi, R. Agrawal, and A. Gupta. *On computing the data cube.* Technical Report RJ10026, IBM Almaden Research Center, San Jose, CA, 1996.

[13] S. Sarawagi, R. Agrawal, and N. Megiddo. *Discovery-driven Exploration of OLAP Data Cubes.* In Proc. International Conf. of Extending Database Technology (EDBT'98), March 1998, 168-182.

[14] Y. Sismanis, A. Deligiannakis, N. Roussopoulos, and Y. Kotidis. *Dwarf: Shrinking the PetaCube.* ACM SIGMOD international conference on Management of data. Madison, Wisconsin, USA. 2002.

[15] A. Shukla, P. Deshpande, J.F. Naughton, and K. Ramasamy. *Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies* VLDB 1996, 522-531.

[16] *TPC BENCHMARK^{TM} D (Decision Support) Standard Specification,* Revision 1.3.1, http://www.tpc.org.

[17] W. Wang, J. Feng, H. Lu, and J. Xu Yu. *Condensed Cube: An Efficient Approach to Reducing Data Cube Size.* Proceedings of the 18th International Conference on Data Engineering, 2002, 155-165.

[18] Y. Zhao, P. M. Deshpande, and J.F. Naughton. *An Array-Based Algorithm for Simultaneous Multidimensional Aggregates.* Proc. of the 1997 ACM-SIGMOD Conf., 1997, 159-170.